VLSI implementation of public-key encryption algorithms

G.A. Orton*, M.P. Roy**, P.A. Scott*,
L.E. Peppard* and S.E. Tavares*

* Department of Electrical Engineering
Queen's University
Kingston, Ontario, Canada K7L 3N6

** Bell Northern Research Ltd.
P.O. Box 3511, Station C
Ottawa, Ontario, Canada K1Y 4H7

Abstract. This paper describes some recently successful results in the CMOS VLSI implementation of public-key data encryption algorithms. Architectural details, circuits, and prototype test results are presented for RSA encryption and multiplication in the finite field $GF(2^m)$. These designs emphasize high throughput and modularity. An asynchronous modulo multiplier is described which permits a significant improvement in RSA encryption throughput relative to previously described synchronous implementations.

A.M. Odlyzko (Ed.): Advances in Cryptology - CRYPTO '86, LNCS 263, pp. 277-301, 1987.

1. Introduction

The RSA algorithm provides a well known, secure implementation of a public-key cryptosystem [1,2,3]. The arithmetic operations required are exponentiation and modulo reduction involving numbers represented by several hundreds of bits. A VLSI approach is justified but presents challenging problems in terms of control generation and distribution circuitry, minimization of storage register size and achieving an adequate throughput rate. Rivest has given a recent review of other attempts to design an RSA chip [4]. Kochanski [5] has described a cascadable chip which implements 32-bit operations on each chip at a rate of 5kbits/sec for 512-bit encryption; however, it appears that considerable redesign is required to compress the implementation to one or a few chips. CYLINK has recently introduced a chip which can perform 512-bit encryption at 6.4kbits/sec in 2um CMOS [6]. A faster design is currently under development at Sandia National Laboratories [7], which uses delayed carry adders to avoid carry propagation delay. This approach is said to be capable of 25kbits/sec in 2um CMOS but has added complexity due to the difficulty of performing comparisons, storage of two K-bit numbers for intermediate results, where K is the number of bits in the modulus, and conversion of the result from the delayed carry representation to binary. Also, the MSB of the modulus must be justified (the message is shifted equally) and the ciphertext returned to LSB justification at the end of the encryption and, as well, the modulo multiplication results need to be shifted 11 bits.

In this paper we describe a bit-slice architecture which incorporates the RSA control functions in the slice along with the arithmetic (modulo multiplication) functions. Registers longer than the modulus are avoided using concurrent modulo reduction. A 32-bit prototype has been fabricated in 3um CMOS and successfully tested. Based on test results and simulations, a throughput rate of 1kbits/sec should be possible for 512-bit encryption with a 2um CMOS process.

In an effort to substantially improve the throughput rate of the bit-slice implementation, a new bit-slice design has been developed employing asynchronous (self-timed) ripple adders [8]. With a small penalty in extra control circuitry, an increase in throughput of up to 40 times can be obtained. A 22-bit prototype in 3um CMOS has been successfully fabricated and tested and a 64-bit version is currently being fabricated.

Multiplication in the finite field $GF(2^m)$ is employed in several data encryption algorithms as well as other areas of communications [7,9]. Recent work has shown that cryptographic algorithms based on arithmetic in $GF(2^m)$ require very large values of m for security [10,11]. In particular, values of m in the range of 1500 bits are recommended. However, for large m, efficient VLSI implementation of the multiplication function requires careful algorithm design to provide modularity and concurrency as well as simplified control requirements. A new multiplication algorithm will be described along with a suitable bit-slice VLSI architecture [12]. Test results from an 8-bit prototype will be presented.

2. Modulo multiplication algorithms

The RSA encryption and decryption transformations involve exponentiation and modulo reduction of a text data block possibly of several hundred bits. The arithmetic process involved is modulo multiplication which requires addition, subtraction and shifting.

Brickell [7] and Blakely [13] have proposed modulo multiplication algorithms in which multiplication is performed concurrently with modulo reduction. This differs from the algorithms used by Rivest

[14] and Simmons and Tavares [15] where multiplication of two K-bit numbers is first performed and then the resulting 2K-bit number is modulo reduced. The maximum word length is (K+1) bits using concurrent modulo reduction. Concurrent algorithms save storage space, reduce adder carry propagation time and require fewer clock periods. Only algorithms of this type will be considered in the remainder of this section.

All of the concurrent algorithms which restrict number lengths to (K+1) bits perform multiplication in one of two ways. The most familiar way of multiplying two numbers is to add shifted versions of the multiplicand or zero depending on the value of the multiplier bits. An example of this technique is shown below in Example 1(a). The second way involves adding the multiplicand or zero to the running total and shifting the running total, as shown in Example 1(b).

Example 1: Binary multiplication

(a)	(b)
1010.	1010.
x1101.	x1101.
1010.	1010.
00000.	1010.
101000.	0000.
+ 1010000.	+ 1010.
10000010.	10000010.

Most techniques of modulo reduction rely on adding some positive or negative multiple of the modulus. With the following concurrent modulo reduction algorithms, the number being reduced is smaller in magnitude than twice the modulus. The modulus is either added or subtracted to reduce the absolute value of the number below the magnitude of the modulus. Modulo reduction can also occur indirectly. An example of indirect modulo reduction of the running total is to first add or subtract the modulus from another number such as the intermediate product (IP) and then add the adjusted IP to the running total. With the two methods of performing multiplication illustrated in Example 1, a variety of methods for concurrent modulo reduction can be employed all of which must prevent overflow by finishing with a number less in magnitude than the modulus. The algorithms operate correctly with starting values less in magnitude than the modulus, so if overflow occurs, the magnitude would continue to increase in subsequent periods. The conditions for concurrent modulo reduction, without overflow, are summarized below.

> If a number, A, which is less in magnitude than the modulus, n, is multiplied by 2 or added to another number, B, which is also less in magnitude than the modulus, the intermediate result can be modulo reduced in the time for one addition. In the case of a positive intermediate result the modulus is subtracted and in the case of a negative intermediate result the modulus is added.

i.e.	if $0 \leq A < n$ then $2A-n < n$ and $2(-A)+n > (-n)$	if $0 \leq A < n$ and $0 \leq B < n$ then $A+B-n < n$ and $(-A)+(-B)+n > (-n)$

A number of useful modulo multiplication algorithms will now be discussed.

Algorithm A

A flow graph of the algorithm is shown in Fig. 1 which is a modification of Blakely's algorithm [13]. Multiplication is done by shifting the intermediate product (IP) and modulo reducing if the IP is greater than the modulus [16]. Then, if the multiplier bit is a 1, the IP is added to the running total. After this, the running total is modulo reduced if necessary. Both the IP and the running total are always positive. A disadvantage of this algorithm is that the running total may require two consecutive additions per multiplier bit.

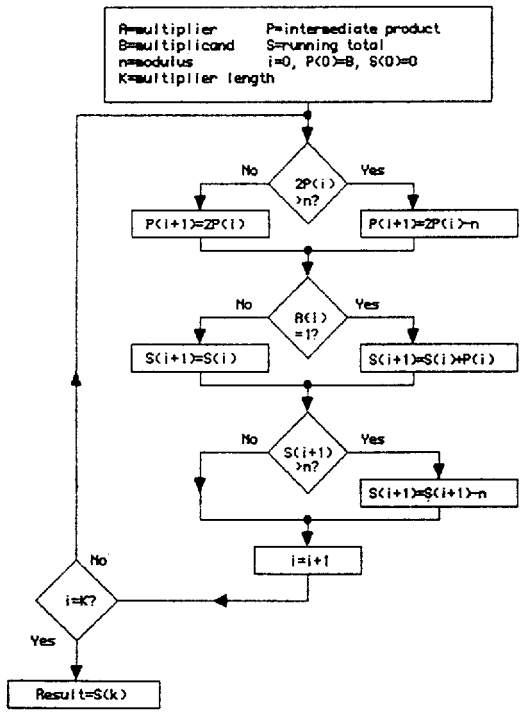

Fig. 1. Modulo multiplication algorithm A. Three adders are used with an average of 1.5 addition phases per multiplier bit.

Algorithm B

A concurrent modulo multiplication algorithm was suggested by Simmons and Tavares [15] which uses multiplication with the running total multiplied by two each period as shown in Fig. 2. A normal cycle starts with the running total being multiplied by 2, followed by adding the IP. Then the running total is modulo reduced using an add/subtract scheme. However, overflow occurs because the combination of multiplication by 2 followed by addition of the multiplicand cannot always be modulo reduced with one addition or subtraction. A necessary modification is to add a negative IP if the running total is positive. This negative IP is generated during the 1st period by adding the positive multiplicand to the running total and then subtracting the modulus to produce a negative result. The negative IP is then stored in a separate register for future use. With this method the final result must be adjusted positive by adding the modulus if necessary. A maximum of one period is required for this

step. Algorithm B requires only 2 adders but, as with algorithm A, two consecutive additions may be required per multiplier bit.

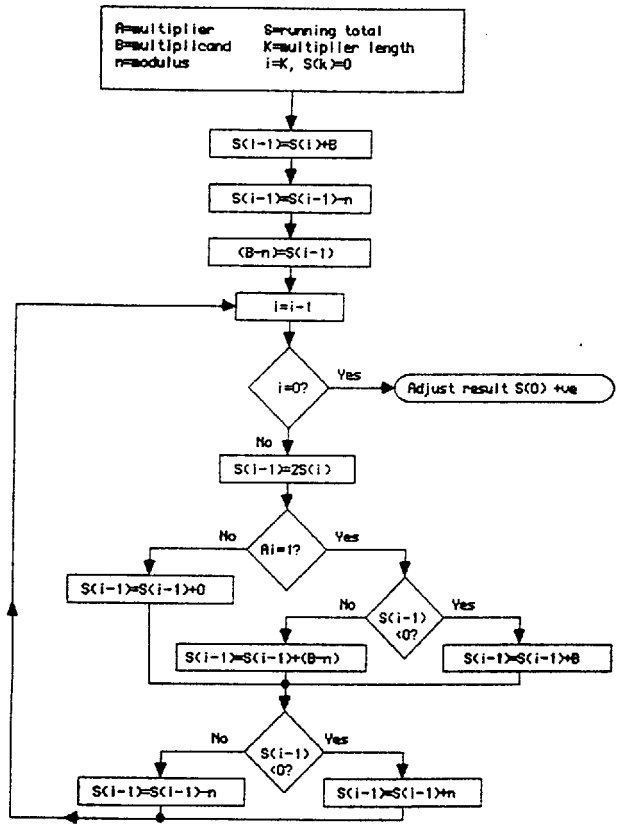

Fig. 2. Algorithm B for modulo multiplication. Two adders are used with an average of 1.5 addition phases per multiplier bit.

Algorithm C

It was thought desirable to consider a modulo multiplication algorithm which would perform all additions during one addition phase per multiplier bit and with a reduced number of adders. An algorithm which uses only 2 adders, which operate concurrently, is algorithm C shown in Fig. 3. In this algorithm, the running total is multiplied by 2 each period. Then if the multiplier least significant bit (MLTlsb) is 0, the running total is modulo reduced. If the MLTlsb is 1, two additions are performed and one of the results is selected as the new modulo reduced running total. This algorithm requires more area because four intermediate products, P, P+n, P-n, and P-2n, need to be first generated then stored. Due to the increase in area required for algorithm C, it was not considered further.

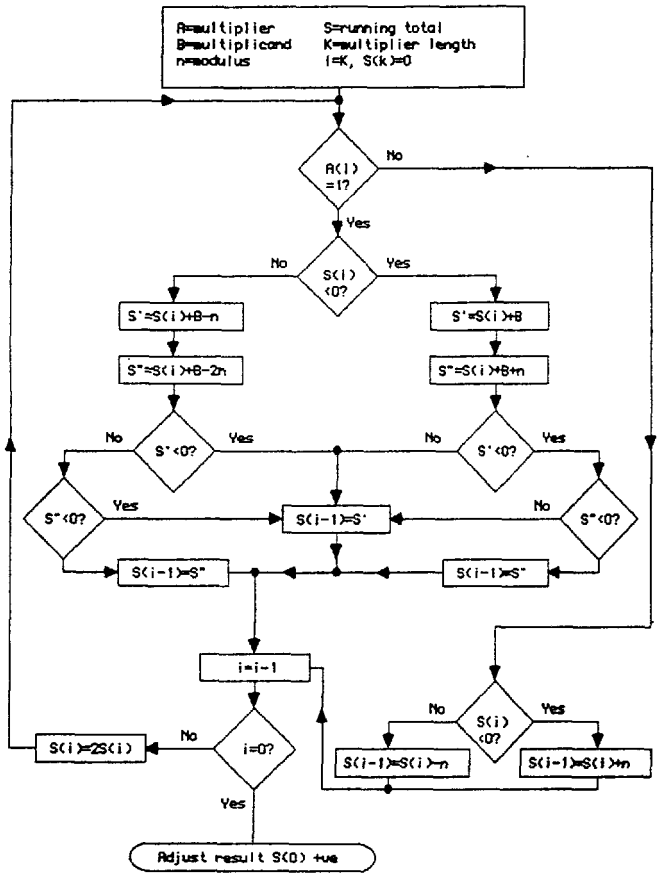

Fig. 3. Algorithm C for modulo multiplication. Two adders are used with one clock phase per multiplier bit.

Algorithm D

Algorithm D performs three additions in a single phase per multiplier bit as shown in Fig. 4. Asymmetrical clock phases are generated with a shortened phase used to set up the adder inputs. As with algorithm A, the previous IP is multiplied by 2 each period. Algorithm D differs in running total generation. If the running total is positive, a negative IP is added to the running total. On the other hand, if the running total is negative, a positive IP is added. This keeps the running total from overflowing and allows it to be generated in one step. The positive IP is generated exactly the same as the IP in algorithm A. At the same time as the positive IP is modulo reduced, twice the modulus is subtracted from twice the previous positive IP. This results in an IP between 0 and -2n. After K periods, where K is the number of bits in the modulus, the multiplication is finished but the running total may need to be adjusted positive by adding the modulus. This takes a maximum of two periods. Algorithm D uses three adders with 1 concurrent addition per multiplier bit and provides a useful compromise between speed and area.

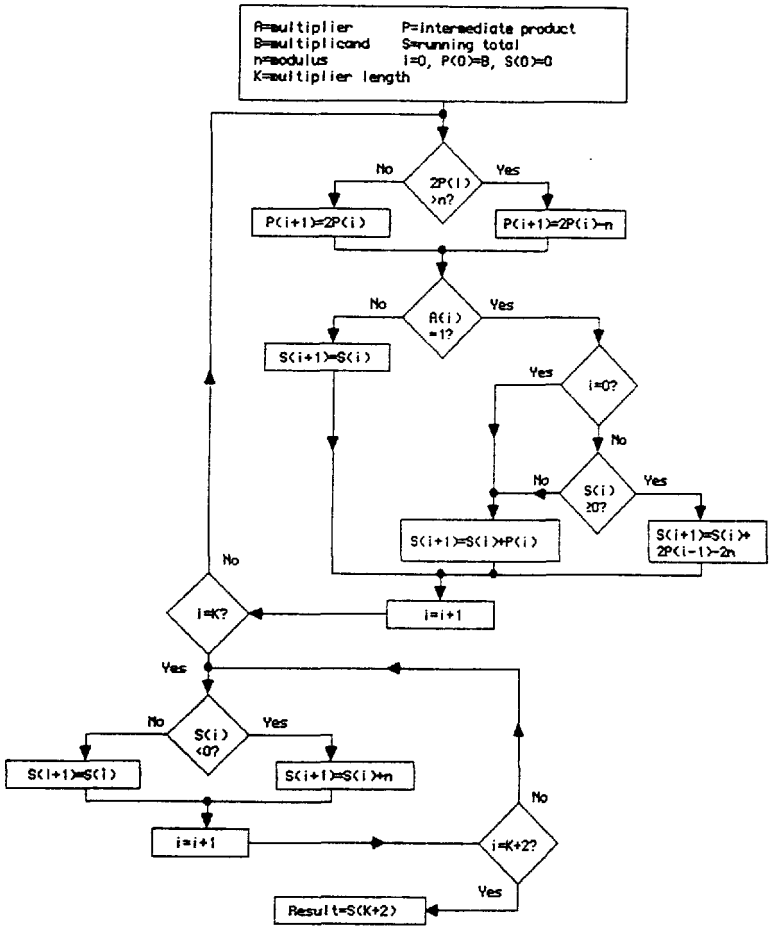

Fig. 4. Modulo multiplication algorithm D. Three adders are used with one phase per multiplier bit.

Algorithm E

An efficient concurrent modulo multiplication algorithm may be devised using the modified Booth's Algorithm (MBA), where the multiplier is shifted two bits at a time. Two consecutive additions per clock period would be required, but the number of clock periods would be reduced by half. This algorithm will be faster if the constant circuit delays in a period are larger than the average addition time which would be the case with a fast adder. In the case of the adder to be described later, approximately a10% to 15% increase in area would result along with 50% improvement in speed compared to algorithm D.

The modified Booth's Algorithm (MBA) is frequently used to improve the speed of multipliers [17]. Through encoding of the multiplier bits, the number of intermediate products to be added is reduced by half. Booth's Algorithm works by skipping over any contiguous string of all 1's or all 0's. A string of all 0's does not require any IP's to be added, but a string of 1's requires an addition and a subtraction. For example, if the multiplier is 11100, (100000 X multiplicand) is added and (10 X multiplicand) is subtracted. The MBA looks at

the three least or most significant multiplier bits at a time depending on the direction that the multiplier is being shifted and shifts the multiplier by 2 bits each clock period. The intermediate products are multiplied by 0, ±1, and ±2 before accumulation as shown in Table 1.

Table 1. Encoding of multiplier for the modified Booth's Algorithm. The centre bit of the three bits being encoded is referred to as the i_{th} bit.

Multiplier bits i+1	i	i-1	Factor of IP accumulated	Operation
0	0	0	0	no string
0	0	1	+1	end of string
0	1	0	+1	a string
0	1	1	+2	end of string
1	0	0	-2	beginning of string
1	0	1	-1	-2+1 = -1
1	1	0	-1	centre of string
1	1	1	0	middle of a string

Algorithm E is diagrammed in Fig. 5 and uses a total of four adders with two adders operating in each phase of a two phase clock. The intermediate products are generated as in algorithm A, but multiplied by 0, ±1, or ±2 before accumulation. Two positive IP's are generated each period consecutively, corresponding to 2 and 4 times the previous IP. Each IP is calculated by shifting the previous IP by 1 bit and subtracting the modulus if necessary. Each period, the appropriate IP is selected, inverted if a negative IP is required and added to the running total. The running total is then modulo reduced by either adding or subtracting the modulus. Two additions are performed each period to generate the IP's and two additions are used to generate the running total. An algorithm which uses fewer additions could be devised at the expense of more memory.

Comparison of modulo multiplication algorithms

The algorithms presented in this section employ concurrency of multiplication and modulo reduction to improve the bit throughput rate. A comparison of six modulo multiplication algorithms is given in Table 2. The selection of the "best" algorithm depends on system parameters such as the delay required for additions relative to constant circuit delays, the availability of non-symmetric clock phases, asynchronous timing and memory. Algorithm D was chosen for implementation because it is almost as fast as algorithm C and occupies about the same area as algorithm A. Algorithm E has only been considered recently. Algorithms A and B are closely matched in speed and area. Long addition times relative to constant delays result in algorithms D and E operating at the same speed, while short addition times make algorithm E twice as fast. With the pulse-timed adder to be described in section 4, the constant circuit delays are at least twice as large as the average addition time, which would make algorithm E at least 50% faster than algorithm D.

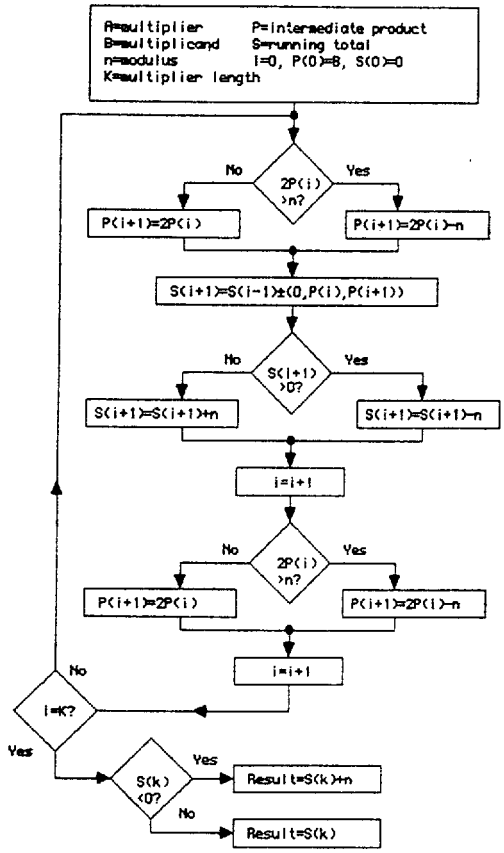

Fig. 5. Modulo multiplication algorithm E. Four adders are used with an average of 1.75 clock phases and the modified Booth's Algorithm.

3. RSA implementation

Architectural aspects

Modulo multiplier architecture. A bit slice architecture for algorithm D is shown in Fig. 6. With a fast adder, communication delays become more significant. Signal flow within the bit slice is less time consuming than the propagation of signals, such as clock signals, MLTlsb, START, ADDER1 carryout, and BEGIN which must be sent to all slices. A completion signal generator subsystem (CSG) is added if pulse-timed adders are used (described in section 4).

A sum term generator controller (STGC) is required to select the input to the running total adder. This subsystem is a 4 to 1 multiplexer, which selects from Cn, Gnd, IP, and (2IP(i-1)-2n) as shown in Fig. 6. The STGC is controlled by logic outside the bit slice which has inputs: MLTlsb, SSRsign, start, phi2, and K or K+1. The signals K or K+1 are from the shift register counter which flags the multiplier to adjust the result positive.

Algorithm	Number of K bit adders	Number of addition phases /multiplier bit	No. of periods per MM	No. of extra registers	Maximum bit rate	Comments
Simmons et.al.	1,(2K bits)	1	2K	0	~20kb/s	Slow, large, and complex.
A	3	Average 1.5	K	0	~35kb/s	Simplest control logic.
B	1 or 2	Average 1.5	K+1	1	~35kb/s	Simplest signal flow in bit-slice.
C	2	1	K+3	3	~42kb/s	Large with extra storage registers.
D	3	1	K+2	0	~40kb/s	Some control logic required outside bit-slice array.
E	4	2	K/2	0	~60kb/s	15% larger than algorithm D.

Table 2. Comparison of modulo multiplication (MM) algorithms.

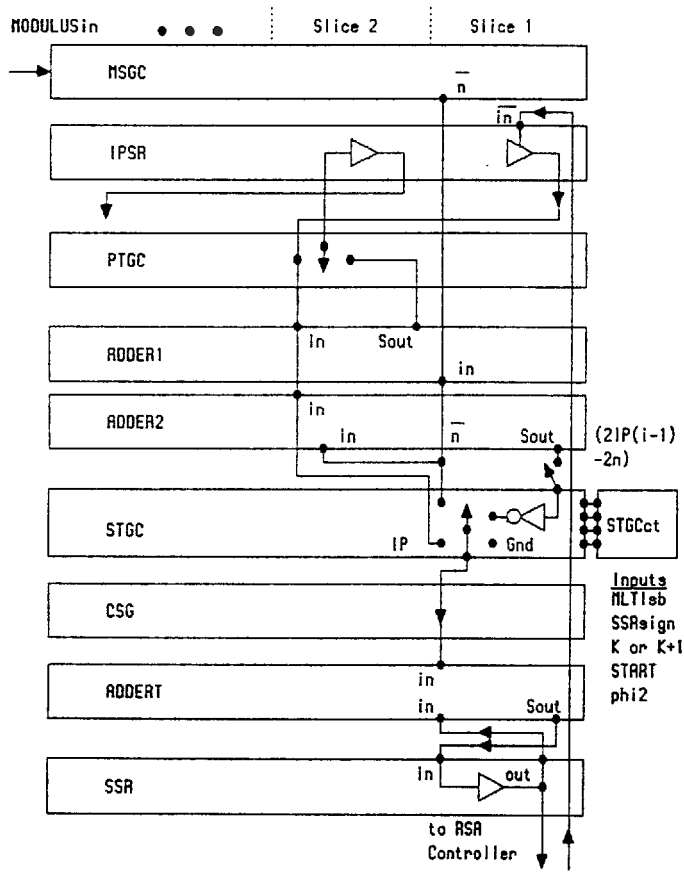

Fig. 6. A bit-slice architecture for algorithm D.

RSA architecture. Three slices of a new bit-slice RSA architecture which includes the implementation of modulo multiplication algorithm D described above is shown in block diagram form in Fig. 7. Modulo exponentiation is performed as a series of modulo multiplications. The message is repeatedly squared and modulo reduced and a running-product modulo multiplication is performed if the corresponding exponent bit is a one. Modulo multiplications are carried out by the subsystems in the upper half of the bit slice while the RSA control functions are implemented in the lower half. The same architecture can be used with other modulo multiplication algorithms. Input and output of data is synchronous and concurrent.

A general purpose register (STRSR) acts as a shifting or storage register with its function determined by control logic outside the bit slice as shown in Fig. 8. The use of this dual purpose subsystem considerably reduces the number of custom subsystems required. The circuitry is compact because control logic for the gates is outside the bit slice. This saves area because one set of control logic is used for all slices. Control logic delay is not a factor since the storage operations are not speed limiting. Standard cells would be suitable for the control logic outside the bit slice. For RSA operation, the message is shifted into both the square term storage

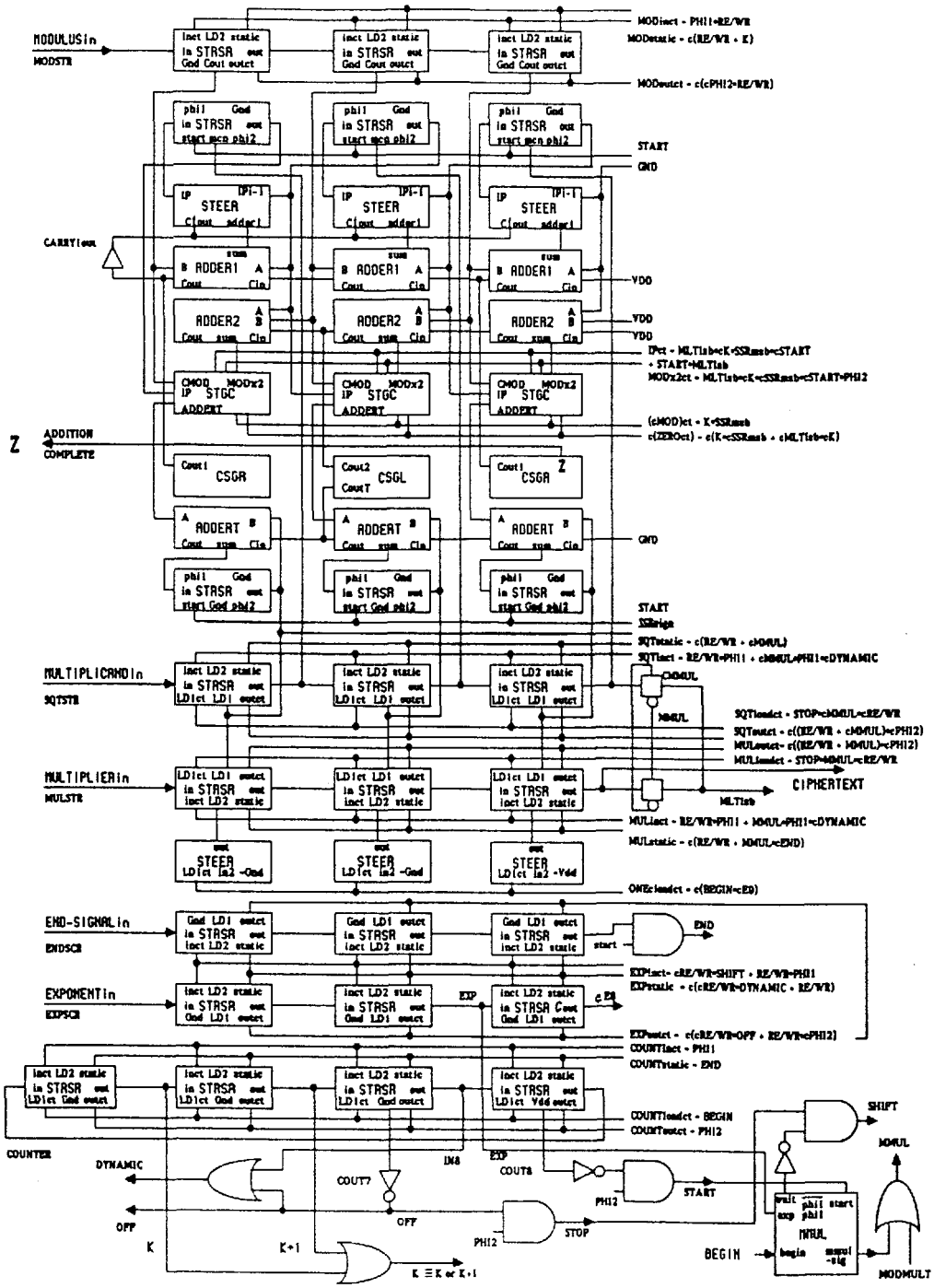

Fig. 7. Bit-slice architecture block diagram (3 slices).

register (SQTSTR) and the multiplication running total storage register (MULSTR). A single modulo multiplication can be performed by loading the multiplicand into SQTSTR and the multiplier into MULSTR.

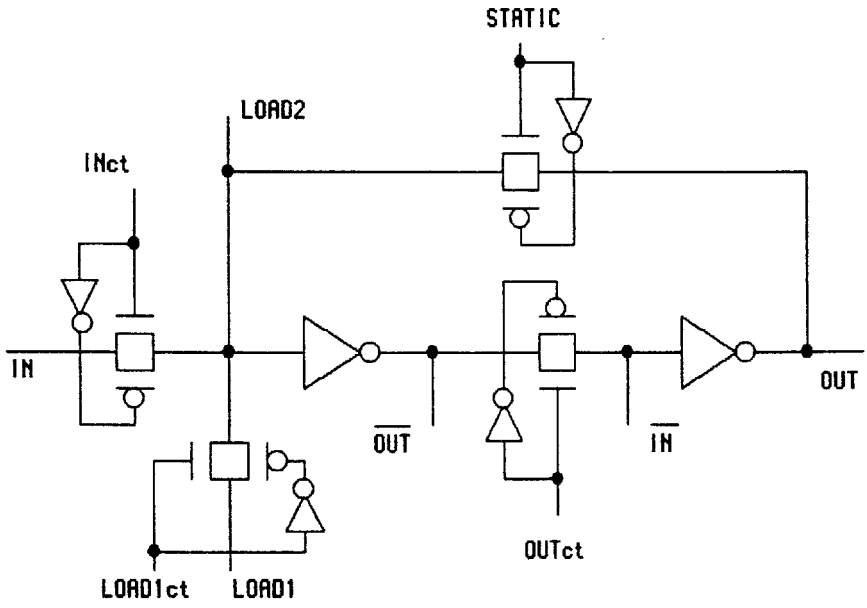

Fig. 8. A multiple function data register (STRSR).

Modulo multiplications are timed with a shift register counter in the bottom row of Fig. 7. The completion of an RSA encryption transformation is set by the END-SIGNAL which allows for exponents of different lengths. The end and exponent registers (ENDSCR and EXPSCR) shift after 1 or 2 modulo multiplications, depending on the exponent bit. Several external control signals are needed: BEGIN starts the encryption; MODMULT sets the chip for a single modulo multiplication; EXT sets the chip to external synchronous timing for data I/O or synchronous testing. The total number of pins required is about 12 depending on the actual application.

Asynchronous operation of the adders can be accommodated using a completion signal generator subsystem (CSG). Fig. 7 shows a pair of CSG subsystems, CSGleft and CSGright which detect all three carry outputs every second slice. For synchronous operation, the CSG subsystems are not required.

Asynchronous aspects

A self-timed adder. Several synchronous adders were considered such as carry lookahead, carry select, and the binary lookahead carry adder [18]. These adders had disadvantages such as high area, irregular layout, slower speed, or the difficulty of providing non-symmetric synchronous clock phases. Past approaches to self-timed adders have been speed independent or Muller circuits which use double rail logic. The disadvantages of this method are slower carry propagation and several times greater implementation area. Pulses have been used successfully to time asynchronous operations, such as asynchronous access of stored state registers [19]. When access is

requested, an edge detector / pulse generator circuit forms a pulse to time the operation.

A new pulse-timed adder which borrows ideas from Hayes [19], is shown in Fig. 9 in which carry propagations are detected in a precharged ripple adder. Carry outputs are reset to 0 during the precharge phase, so only propagations of 1 have to be detected. An edge detector / pulse generator circuit provides enough delay for the carry signal to propagate to the next pulse subsystem. A pulse subsystem is only half as large as a low area precharge adder slice. Pulses are combined to create the completion signal with a single active load pullup NOR gate.

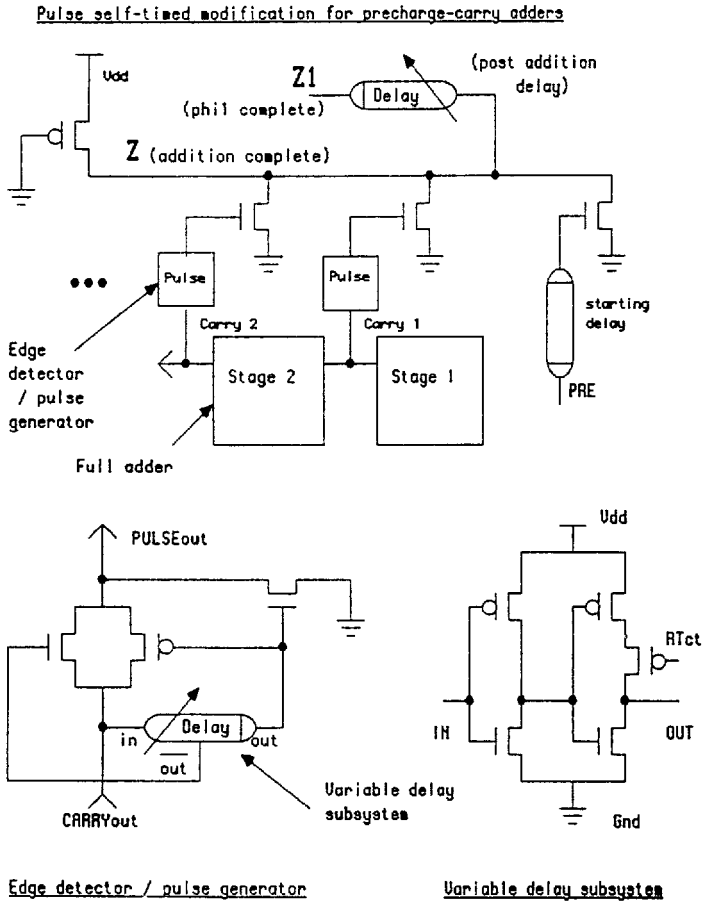

Fig. 9. Self-timed adder circuit details.

Several features of the adders make this scheme feasible. Overlap of the pulses prevents premature generation of the completion signal. The pulses are made several times wider than necessary since the resulting delay at the end of the additions is absorbed by subsequent RSA operations. The mean of the maximum number of consecutive carries in a 512 stage adder is only 8.2 with a variance of 3.3 (almost 60 times faster than

a ripple adder alone, on average!). The probability of less than 5 carries is negligible so a starting delay equivalent to 4 carries can be provided to overlap the first pulses. A pulse subsystem is not required every slice and all three adders calculating during the same phase can signal with the same NOR gate pulldown.

Optimization of the adder speed must be balanced against area considerations. A pulse subsystem every slice is the fastest on average. However too many pulldowns slow down the large NOR gate. The number of pulldowns can be reduced by grouping pulses with smaller NOR and NAND gates first. These variables were adjusted to achieve low area and a reguler layout.

Clocking. In combining the modulo multiplication algorithm of Fig. 4 with the self-timed adder of Fig. 9 in the bit-slice architecture of Fig. 7, several control and timing considerations must be addressed.

The clock has to be capable of switching between asynchronous and synchronous timing to allow synchronous I/O and testing. Also, when the RSA encryption is finished, the clock should stop with the ciphertext safely in a storage register. These functions are implemented with random logic as shown in Fig. 10. A Muller C element is used to prevent the PHI2 clock signal from going low until the PHI1 clock phase has risen to prevent a race condition. This allows the PHI2 falltime to be set to the minimum value and only the risetime of the variable delay elements have to be adjustable.

Driver delays form a significant part of the clock period. Delays for generating some control signals cannot be avoided but the delay of the clock drivers can be largely prevented from adding to the clock period. Most of the falltime of clock phases does not contribute to the clock period because the clock phase widths can be externally adjusted. Also the non-overlap time can be externally minimized as shown in Fig. 10. In the 8 and 24 slice implementations, the inverted driver outputs, cPHI1 and cPHI2, were fed back to the clock controller rather than delayed versions of cPHI1sig and cPHI2sig. This guarantees that there is sufficient non-overlap time but it is not adjustable and results in an approximately 30% larger clock period.

For some parts of the circuit, considerable area would be required to generate completion signals logically. Delay elements were used instead with active load resistors to time these circuits. Active loads can provide sufficient delay in a small area and can be controlled by an external DC voltage (RTct in Fig. 9). The new data rate control scheme employs a single pin to control all delay elements. An intermediate DC voltage is first selected, say, 2.5 Volts. Then the gate aspect ratio of each active load is chosen to provide the expected circuit delay. During testing, the DC voltage is reduced to find the maximum operating rate (similar to finding the maximum clock rate of a synchronous chip). The accuracy and stability of the active loads can be improved by increasing the gate length and width while keeping the gate aspect ratio constant. This asynchronous timing method has the advantages of rate controllability, low area, elimination of global clock distribution, and allows different processes to be timed at their own rate. Lastly, it uses only 1 pin. Correct chip timing is ensured since the delay of each variable delay element can be increased arbitrarily.

Synchronization failure can occur when gating an asynchronous signal to a synchronous system. Only latching the END signal is prone to this type of failure. In an encryption environment, a host processor would periodically sample the END signal and there is a small probability that a metastable state would be detected. Increasing the settling time rapidly decreases the failure rate to an acceptable level [20]. The required settling time is negligible compared to the encryption time.

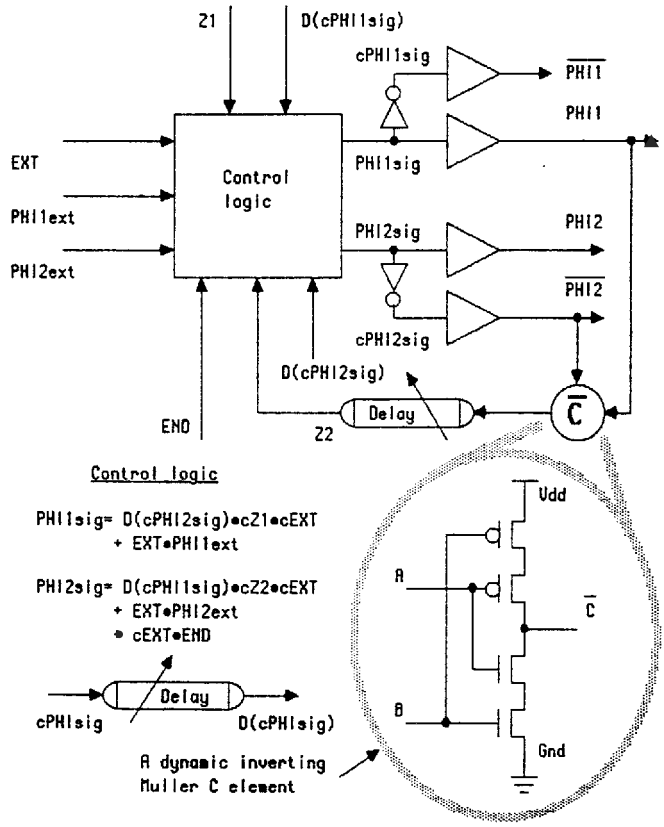

Fig. 10. Clock controller for the RSA chip with algorithm D and asynchronous adders. The non-overlap time is adjustable.

Implementation

Output speed limits. Output driver current will often limit a synchronous clock rate while an asynchronous clock is not I/O limited since there is no I/O during the RSA transformation. The estimated average asynchronous clock rate in 2um CMOS is as high as 30Mhz. This advantage of asynchronous timing will become more pronounced as processes scale down further. Circuit speed scales down as A^2 [21], where A is the scaling ratio, if the power supply voltage is held constant. However, the driver speed scales down as A/ln(A) [22]. Asynchronous techniques could also be applied to any synchronous algorithm to allow the RSA encryption to proceed faster than the I/O speed.

The data rate of algorithm D. Algorithm D requires one clock phase for addition plus a shorter phase to set up the adder inputs. The throughput rate is effected by the on-chip communication delays which are hard to estimate accurately since they depend on the particular manufacturing process. Estimates based on SPICE simulations of signal propagation delays can be made. A calculation of the bit rate for algorithm D with the pulse-timed adder yields a rate of 40kbits/sec. Details of this calculation are provided in Appendix A. This corresponds to an average asynchronous clock rate of 30Mhz. A slower synchronous clock rate would be used for I/O, but a negligible number

of clock periods are required for I/O. At the expected clock rates, small variations in the circuit speed have a large effect on the throughput rate, but a conservative estimate of 30kbits/sec appears reasonable.

A synchronous bit-slice implementation. Fig. 11 is a photomicrograph of a 32-bit prototype chip executed in 3um CMOS. Algorithm A has been used for modulo multiplication. A different architecture than that shown in Fig. 7 is employed in the synchronous implementation, which simplifies the control logic external to the slice at the expense of more custom registers. The bit slices run horizontally and are comprised of 14 subsystems which implement modulo multiplication, exponentiation and storage functions. Input and output data flow is serial which minimizes the total pin count. This chip has been tested and shown to correctly perform RSA encryption (or decryption) at a synchronous clock speed of 200kHz, which corresponds to a rate of 4kbits/sec. The synchronous, pre-charged adder delay per bit has been measured to be 8 ns in 5um CMOS from which a throughput rate of 1kbits/sec for 512-bit encryption is predicted for a 2 micron CMOS process. For 3um CMOS and 32-bit encryption, a rate of 5MHz and 100kbits/sec encryption is predicted. The low measured speed is difficult to explain since a 7-bit prototype in 5um CMOS was found to operate at 2MHz. More samples are being bonded for testing which may indicate if process parameter variations are involved.

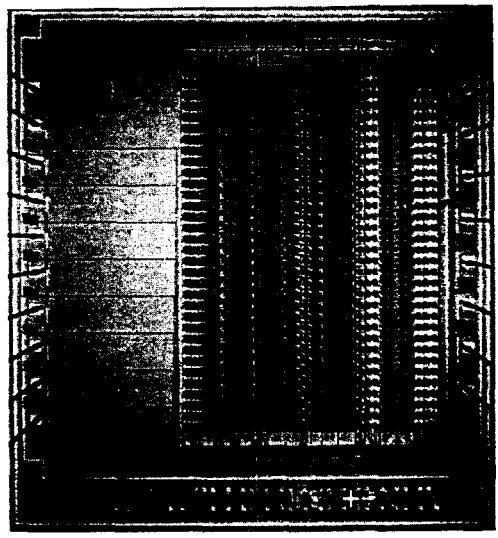

Fig. 11. Photomicrograph of synchronous 32-bit RSA prototype implemented in 3um CMOS.

Asynchronous implementation. Fig. 12 is a photomicrograph of a 24 slice implementation of an asynchronous RSA design based on algorithm D, the architecture described in Fig. 7 and the pulse-timed adder. The data analyser display for a 22 bit encryption is shown in Fig. 13. Input and output of data are overlapped, so both input and the previous output can be seen at the same time. Both inputs and outputs start least significant bit first.

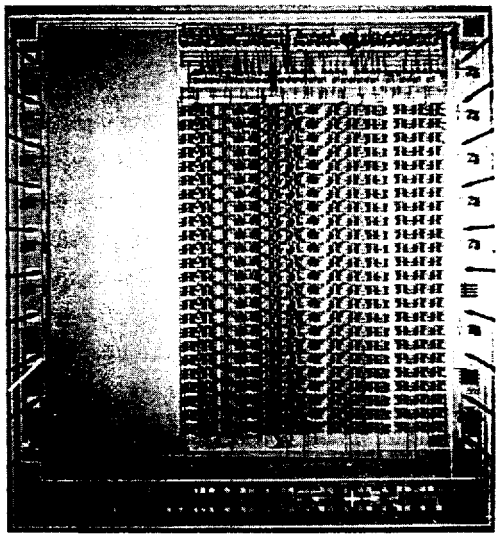

Fig. 12. Photomicrograph of asynchronous 22-bit RSA prototype implemented in 3um CMOS.

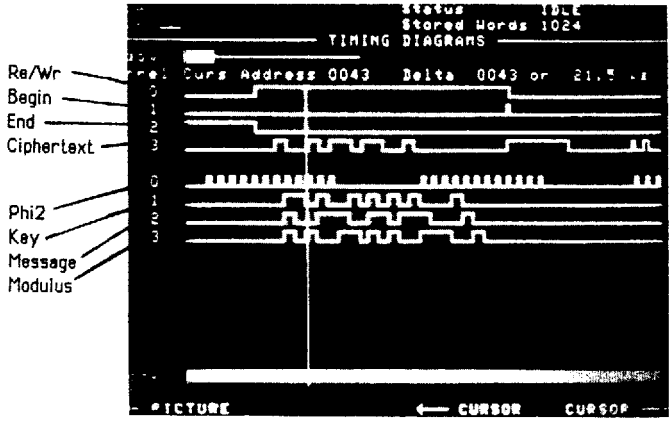

Fig. 13. Data analyzer display for a 22-bit computation: $584{,}932^{283{,}948} \bmod (1{,}283{,}476) = 19{,}876$. Due to the slow sampling rate PHI2ext appears to stay at 0 sometimes. A single input set was cycled, so this ciphertext corresponds to this input set.

The average asynchronous clock rates of these designs provide a good indication of the accuracy of the speed extrapolations made in Appendix A. In 2um CMOS for 512-bit encryption, the estimated optimized throughput was 40kbits/sec with an average clock rate of 30MHz. In 5um CMOS for an 8 slice prototype, the average clock rate was found to be 3Mhz and the encryption rate was 300kbits/sec, while in 3um CMOS for 24 slices the average clock rate was found to be 5MHz and the encryption rate 150kbits/sec. The estimated optimized average clock frequencies for these processes were 6MHz for 5um CMOS (8 slices) and 13MHz for 3um CMOS (24 slices). Additional samples are being bonded to determine if the slower than predicted clock rate is related to process variations. In any case, there are some further steps which can be taken to increase speed, including use of double metalization, so that it seems possible that the predicted performance can be attained.

Work in progress. Expansion of the asynchronous design to perform transforms involving many hundred bits is necessary to verify the speed advantages of the architecture and algorithms described here. The 64-bit chip presently being fabricated in 3um CMOS will help to verify the extrapolations which were made to predict the speed of a 512 bit design in 2um CMOS. A 128-bit version has also been designed and will be fabricated in the near future.

Significant further improvements in the throughput rate of RSA encryption are not likely to come from faster adders. With the asynchronous pulse adder, constant circuit delays take about twice as long as the three concurrent additions. The constant delays which result from signal propagation delays (excluding additions) are difficult to reduce in this style of architecture. Thus, a faster adder could only achieve about a 30% speed improvement at the most. Future improvements may be possible with new architectures.

Higher bit rates can be achieved by interconnecting chips in several patterns. One suggested architecture is a systolic arrangement of modulo multipliers [23]. This design cascades at least K modulo multipliers with a systolic data flow, where K is the number of bits. New systolic arrangements of asynchronous encryption units are faster and can be built with any number of encryption units. Binary tree input distribution, with token ring chip selection is the most efficient and achieves the same performance as a single encryption chip.

4. A multiplier for the finite field GF(2^m)

Arithmetic operations in the finite field GF(2^m) are quite different from ordinary integer arithmetic operations. Addition does not involve carries and is thus easier to perform than integer addition, but multiplication is still a fairly complex and difficult task. Most circuits proposed [9,24] are not suited for use in VLSI systems. They require excessive silicon area, complicated control schemes, complex wire routing, have nonmodular structures, or lack concurrency [12].

The systolic multiplier developed by Yeh, Reed and Truong [25] is suitable for VLSI implementation although it is only moderately compact and has a latency of 2m time units which may be undesirably long for some applications. The implementation of the Massey-Omura multiplier [26] is simpler than the systolic version and operates with a smaller latency, but is less modular and has a circuit structure and operating speed which is dependent on the size of the field.

The architecture to be described here uses an approach similar to the one outlined by Laws and Rushforth [27]. It is modular and therefore easily expanded, compact, and requires few control signals. The multiplication time and latency are m time units.

The algorithm

It is assumed that the reader has a basic knowledge of finite fields. If $A(x) = a_{m-1}x^{m-1} + \ldots + a_1x + a_0$ and $B(x) = b_{m-1}x^{m-1} + \ldots + b_1x + b_0$ are two elements of GF(2^m), then their product, $A(x)B(x) \bmod F(x)$, is $P(x) = p_{m-1}x^{m-1} + \ldots + p_1x + p_0$, where $F(x) = f_{m-1}x^{m-1} + \ldots + f_1x + 1$ is an irreducible polynomial.

The multiplication, A(x)B(x)modF(x), can be expanded by multiplying each term of B(x) by A(x):

$$\begin{aligned} P(x) &= A(x)B(x)\,\mathrm{mod}F(x) \\ &= \{A(x)b_{m-1}x^{m-1}\,\mathrm{mod}F(x) + \ldots + A(x)b_1 x\,\mathrm{mod}F(x) \\ &\quad + A(x)b_0\,\mathrm{mod}F(x)\}\,\mathrm{mod}F(x) \end{aligned}$$

The first term $A(x)b_{m-1}x^{m-1}\,\mathrm{mod}F(x)$ is computed, followed by each successive term which is added to it and the sum reduced modF(x) until all the terms have been used.

If $A = [a_{m-1},\ldots,a_1,a_0]$ is the vector of coefficients of A(x) and similarly for B(x), P(x) and F(x), then this algorithm can be represented by the flowchart in Fig. 14. Element A is added to the intermediate product, P, whenever the current bit of B, b_I, is a 1. F is added whenever the most significant bit (MSB) of P is 1, which indicates that modulo reduction is necessary. These two decisions are carried out simultaneously. If the field is of degree m, then m steps are needed to complete a multiplication.

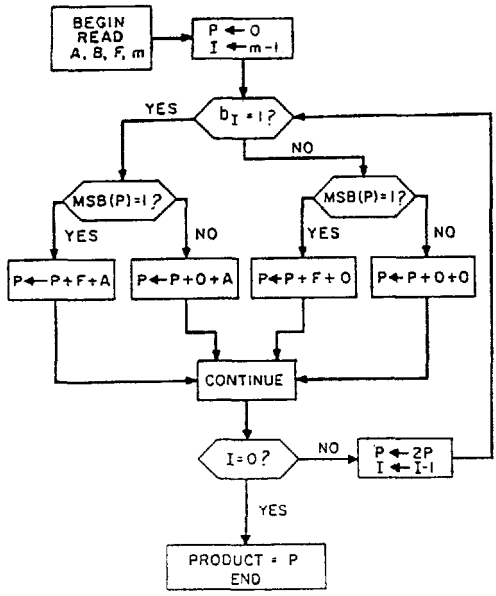

Fig. 14. Flowchart of $GF(2^m)$ multiplication algorithm.

Architecture

The multiplier architecture is shown in Fig. 15 for the field $GF(2^4)$. Registers a_i, f_i and p_i hold A, F and the intermediate product P, respectively. The MSB of F, which is always 1, is actually not used in the

calculation. The state of b_i, latched with a flip-flop, and the MSB of P constitute the two primary control signals. The left shift is performed by loading the output of stage L_i into the product register p_{i+1} of the next stage L_{i+1}. The final product is transferred to the output shift register (OSR) and shifted out serially once the multiplication is complete. Note that the worst case delay path from the f_i register to the OSR is independent of the multiplier size. The number of L_i and register stages is equal to the degree of the field, in this case four.

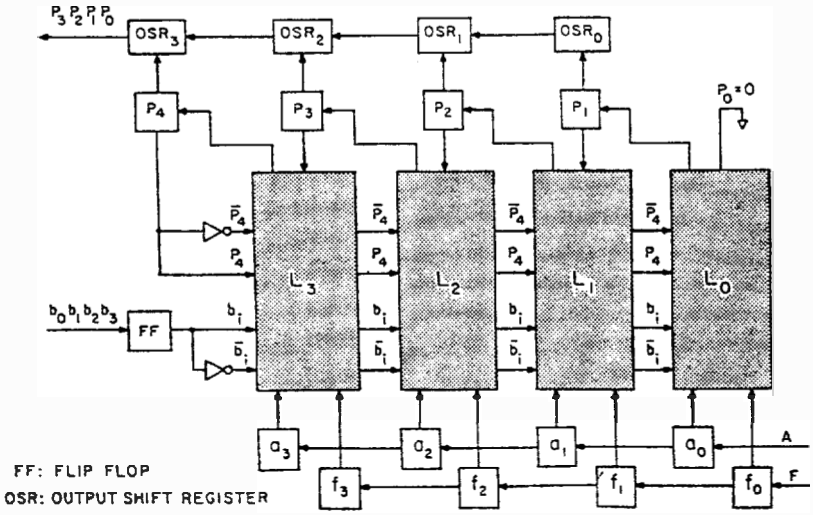

Fig. 15. The multiplier for $GF(2^4)$

Each stage L_i contains one 3-input modulo, two transmission gates and two NMOS transistors, as detailed in Fig. 16. The transmission gates and transistors are configured to perform the AND function (MSB(P) AND f_i, and b_i AND a_i). For example, if $b_i = 1$ then a_i is passed to the adder; otherwise that adder input line is grounded (set to 0).

Implementation and testing

A multiplier for $GF(2^8)$ was implemented using a combination of static and dynamic logic and fabricated in 5 micron CMOS. It was found to be fully functional, capable of operating at speeds up to 7 Mbits/sec [28]. As SPICE simulations predicted, the data rate was limited by the speed of the output pad drivers, not the worst case delay path on the chip. Optimization of the pad drivers should improve the speed by about 30-40%.

Each of the eight slices occupied an area of 185 microns by 1459 microns, of which 10% was allocated to test structures. Subsequent chips have been modified to incorporate a more structured design for testability approach, the Scan Path technique [29]. In addition, the pad drivers and adders were replaced with faster versions. The new slice occupies 23% less area.

This enhanced 8-bit version was submitted for fabrication in 3 micron CMOS in September 1986, along with a 128-bit multiplier. A 512-bit chip, with a total area (multiplier and I/O pads) of 6912 microns by 6980 microns, will be submitted at the end of 1986. From these two larger designs it is hoped that more information about the performance of the algorithm will be obtained.

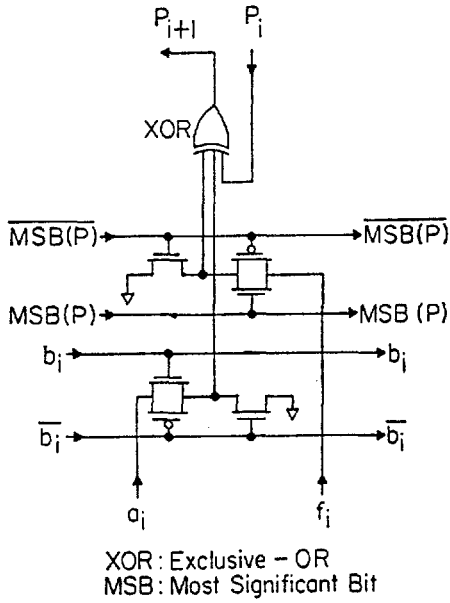

Fig. 16. The circuit for each block Li shown in Fig. 15.

5. Conclusion

RSA architecture. A 22-bit, 3um CMOS prototype of an asynchronous RSA chip has been fabricated and found to function correctly with a throughput rate of 150kbits/sec. A conservative estimate for the 512 bit encryption rate in 2um CMOS is 30kbits/sec with optimization of the present design and 40kbits/sec with algorithm E. The asynchronous clock rate during encryption is not I/O limited nor is it limited by the clock rate in other components.

Concurrent modulo multiplication algorithms provide the most efficient implementations known for RSA encryption. Multi-adder algorithms such as algorithms D and E are efficiently implemented with the asynchronous pulse-timed adder. Minimization of constant circuit delays is important since they several times larger than the addition time in a clock period.

$GF(2^m)$ multiplier. To evaluate the performance of the finite field multiplication algorithm, an 8-bit prototype has been fabricated in 5um CMOS and tested. It was found to operate correctly for data rates up to 7Mbits/sec. A new 8-bit version with faster adders and pad drivers, and a more structured approach to testing is currently being fabricated in 3um CMOS along with a 128-bit multiplier. A 512-bit chip will be implemented at the end of 1986

and should provide some useful information about large VLSI multipliers.

6. Appendix A: Calculation of the RSA throughput rate with algorithm D

Constant on-chip communication delays in 5um CMOS:
Non-overlap time = 2x10ns (if adjustable)
Signal flow after addition: drive carryout line plus signal flow within bit slice = 30ns
Clock transition time = 2x10ns (crossing threshold only)
phi2: control generation (15ns) plus driving of control lines plus • signal flow within bit slice = 50ns

---

Total = 120ns

1 clock period in a 5um process = TD5 = (Nc +L)Tp + 120ns = 201.2ns

where Nc = average number of carries for an average of 2.5 adders of 500 bits each = 9.6
L = No. of slices separating pulse subsystems of adders = 2
and Tp = carry propagation speed in a 5um process = 7ns/slice

1 clock period in a 2um process = TD2 = TD5•$(2/5)^2$ = 32.2ns

RSA transform execution time = Texe = (the number of modulo multiplications)(the number of periods per multiplication)(the length of a period)

Texe = (1.5•K)•(K+2)•TD2 = .0127 sec

where K = number of bits in exponent and modulus = 512

Bit rate = K/Texe = 40 kbits/sec

7. Acknowledgements

The research reported here was supported in part by strategic grants 60893, 60894 and 61364 and Operating Grants from the Natural Sciences and Engineering Research Council of Canada. VLSI design and testing equipment was provided under the loan program of the Canadian Microelectronics Corp. (CMC). Chip fabrication was carried out by Northern Telecom Electronics Ltd. under the fabrication program of the CMC.

8. References

[1] W. Diffie and M. Hellman, "New Directions in Cryptography", IEEE Trans. Info. Theory, Vol. IT-22 (6), pp. 644-659, Nov. 1976.

[2] R.L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems", Comm. of the ACM, Vol. 21, No. 2, pp 120-126, Feb. 1978.

[3] D. Denning, "Cryptography and Data Security", Reading, Mass.: Addison-Wesley Publ. Co., 1982.

[4] R.L. Rivest, "RSA Chips (Past/Present/Future)", Advances in Cryptology, Proc. of EUROCRYPT 84, pp. 159-165, Springer-Verlag, Berlin, 1985.

[5] M. Kochanski, "Developing an RSA Chip", Proc. of CRYPTO 85, Santa Barbara, CA., Aug. 1985.

[6] CYLINK, "Advance Data Sheet: CY1024 Key Management Processor", CYLINK, 920 West Fremont Ave., Sunnyvale, California 94087, 1986.

[7] E.F. Brickell, "A Fast Modular Multiplication Algorithm with Application to Two Key Cryptography", Proceedings of CRYPTO 82, Santa Barbara, California, pp. 51-60, August 1982.

[8] G.A. Orton, L.E. Peppard, and S.E. Tavares, "A Fast Asynchronous RSA Chip", IEEE Custom Integrated Circuits Conference, Rochester, N.Y., pp. 439-443, May 12-15, 1986.

[9] W.W. Peterson and E.J. Weldon, "Error-Correcting Codes", Cambridge, MA: MIT Press, 1972.

[10] A.M. Odlyzko, "Discrete Logarithms in Finite Fields and Their Cryptographic Significance", Advances in Cryptology, Proc. of EUROCRYPT 84, pp. 225-314, Springer-Verlag, Berlin, 1985.

[11] I.F. Blake, R. Fuji-Hara, R. Mullin and S. Vanstone, "Computing Logarithms in Finite-Fields of Characteristic Two", SIAM J. Alg. Discr. Methods, Vol. 5, pp. 276-285, 1984.

[12] P.A. Scott, S.E. Tavares and L.E. Peppard, "A Fast VLSI Multiplier for GF(2^m)", IEEE Journal on Selected Areas in Comm., Vol. SAC-4, pp. 62-66, January 1986.

[13] G.R. Blakely, "A Computer Algorithm for Calculating the Product AB Modulo M", IEEE Trans. Computers, Vol. C-32, pp. 497-500, May 1983.

[14] R.L. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher", Lambda (Fourth Quarter 1980) pp. 14-18.

[15] D. Simmons and S.E. Tavares, "An NMOS Implementation of a Large Number Multiplier for Data Encryption Systems", Proc. 1983 Custom Integrated Circuits Conf., Rochester, N.Y., pp. 262-266, May 1983.

[16] M.P. Roy, L.E. Peppard and S.E. Tavares, "A CMOS Bit-Slice Implementation of the RSA Public-Key Encryption Algorithm", 1985 Canadian Conference on Very Large Scale Integration, Toronto, Canada, pp. 52-56, November 1985.

[17] S. Waser and A. Peterson, "Real-time Processing Gains Ground with Fast Digital Multiplier", Electronics, pp. 93-99, September 29,1977.

[18] N. Weste and K. Eshraghian, "The Principles of CMOS VLSI Design: A Systems Perspective", Addison-Wesley, 1985.

[19] A.B. Hayes, "Self-Timed IC Design with PPL's", Third Caltech Conference on VLSI, Computer Science Press, Inc., Rockville, Maryland, 1983, pp. 257-274.

[20] T.J. Chaney and F.U. Rosesenberger, "Characterization and Scaling Of MOS Flip Flop Performance in Synchronizer Applications", Proceedings of the First Caltech Conference on VLSI, 1979.

[21] C.L. Seitz, "Self-Timed VLSI Systems", Proceedings of the First Caltech Conference on VLSI, pp. 345-354, January 1979.

[22] D.R. Brown, "Optimization of On-Chip Input/Output Interfacing Circuitry for VLSI Systems", M.Sc. Thesis, Department of Electrical Engineering, Queen's University, July 1985.

[23] K. Culik II, Jürgensen, K. Mak, "Systolic Tree Architecture for some Standard Functions", Report 140, Dep. of Computer Science, University of Western Ontario.

[24] T.C. Bartee and D.I. Schneider, "Computation with Finite Fields", Inform. and Control 6, pp.79-98, 1963.

[25] C.S. Yeh, I.S. Reed, and T.K. Truong, "Systolic Multipliers for Finite Fields GF(2^m)", IEEE Trans. Comput., vol. C-33, pp. 357-360, April 1984.

[26] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in GF(2^m)", IEEE Trans. Comput., vol. C-34, pp. 709-717, Aug. 1985.

[27] B.A. Laws, Jr. and C.K. Rushforth, "A Cellular-Array Multiplier for GF(2^m)", IEEE Trans. Comput., vol. C-20, pp. 1573-1578, Dec. 1971.

[28] G.A. Orton, M.P. Roy, P.A. Scott, L.E. Peppard, and S.E. Tavares, "New Results in Mapping Data Encryption Algorithms into VLSI", presented at the Fourth Int. Workshop on VLSI in Comm., Ottawa, Ont., June 1986.

[29] T.W. Williams and K.P. Parker, "Design for Testability - a Survey", Proc. IEEE, vol. 71, pp. 98-112, Jan. 1983.